PI-WHISPER on Edge: Real-Time Deployment System Design on NVIDIA Jetson

Piyush Modi

piyushmo@buffalo.edu

Mentors

Amir Nassereldine
amirnass@buffalo.edu

Dancheng Liu
dliu37@buffalo.edu

Advisor

Dr. Jinjun Xiong
jinjun@buffalo.edu

Department of Computer Science
University at Buffalo

May 15, 2026

Abstract

Automatic Speech Recognition (ASR) systems have traditionally relied on cloud infrastructure, raising significant concerns regarding data privacy, communication latency, and offline availability. Deploying robust foundational models like OpenAI's Whisper on edge devices mitigates these issues but introduces acute challenges due to strict limitations on computational resources, memory, and power. This report details the system design and real-time deployment of PI-Whisper on resource-constrained edge hardware, specifically targeting the NVIDIA Jetson and Raspberry Pi platforms. PI-Whisper introduces an ultra-low latency, modular framework that dynamically tailors inference to a speaker's unique traits, such as age, gender, and accent, without retraining the entire model. The deployed architecture utilizes specialized 12-layer Convolutional Neural Networks (CNNs) to predict speaker characteristics from a rapid 3-second audio tensor. These predictions drive a weighted Parameter-Efficient Fine-Tuning (PEFT) adapter that dynamically pools targeted Low-Rank Adaptation (LoRA) profiles on the fly. Using Pi-Whisper, the system achieves up to a 13.7% relative reduction in Word Error Rate (WER) across diverse demographics while adding only 0.18 seconds of inference time. Additionally, by using task-scoped

GPU worker isolation and MJPEG buffering, the system can stream video to the user without interfering with inference. This work demonstrates a highly viable, scalable architecture for inclusive, personalized, and secure on-device speech recognition.

1 Introduction

The proliferation of voice-driven applications has established Automatic Speech Recognition (ASR) as a foundational component of modern computing. However, the heavy reliance of state-of-the-art ASR systems on cloud processing introduces critical vulnerabilities. Transmitting sensitive voice data to remote servers poses severe privacy and security risks, while the inherent network communication introduces latency that degrades the user experience in real-time applications. Furthermore, cloud-dependent systems fail in low-connectivity or offline environments. Transitioning ASR capabilities directly to edge devices offers a definitive solution to these challenges, enabling secure, localized, and immediate speech processing.

Despite the clear advantages of on-device inference, deploying massive, highly parameterized models like OpenAI's Whisper on edge hardware presents formidable technical hurdles. Edge environments, such as the Raspberry Pi or NVIDIA Jetson series, impose strict constraints on processing power, available memory, and energy consumption. Additionally, to be truly effective and equitable, an ASR system must maintain high accuracy across a highly diverse user base. This necessitates an architecture capable of real-time adaptivity to individual speaker characteristics, such as varied accents, genders, and age groups, without incurring the computational burden of retraining the entire model.

This graduation report outlines the practical implementation and system-level optimization of PI-Whisper, an ultra-low latency ASR framework designed to bridge the gap between foundational speech models and edge-device constraints. Building upon the core PI-Whisper research methodology, this project focuses heavily on the robust deployment architecture required to execute these complex models in real-time. Conducted in alignment with ongoing edge computing research, the work details a distributed, asynchronous pipeline optimized for hardware like the NVIDIA Jetson Orin Nano.

The implemented system extracts the first 3 seconds of input audio and routes it through discrete CNNs to identify speaker traits on the fly. These identified traits dictate the dynamic retrieval and concatenation of specific LoRA adapters from a pre-compiled profile library, which are then merged into the Whisper encoder and decoder via a PEFT strategy. By combining this dynamic profile mixing with strict memory management protocols, this deployment successfully realizes the theoretical benefits of PI-Whisper, delivering measurable fairness gains and WER reductions while maintaining the strict latency requirements of real-world edge applications.

Fig. 1 shows the original architecture proposed by the authors of [1]. The paper presented how the PI-Whisper system augments existing ASR frameworks with Low-Rank Adaptation (LoRA) profile libraries and an optional classifier. Our platform uses this to support the determination of a speaker's age, gender, and accent. The platform is adaptable to situations where the speaker might want to provide only one of the 3 characteristics or all of them or none. The software can dynamically handle each of these situations. When the speaker's characteristics are to be determined, a classifier detects the optimal characteristics of the speakers and loads the appropriate LoRA profile for ASR customization. The paper addresses ASR services like adaptivity, incrementality, and inclusivity. It shows that the human voices can be characterized into different characteristic libraries and improve ASR performance.

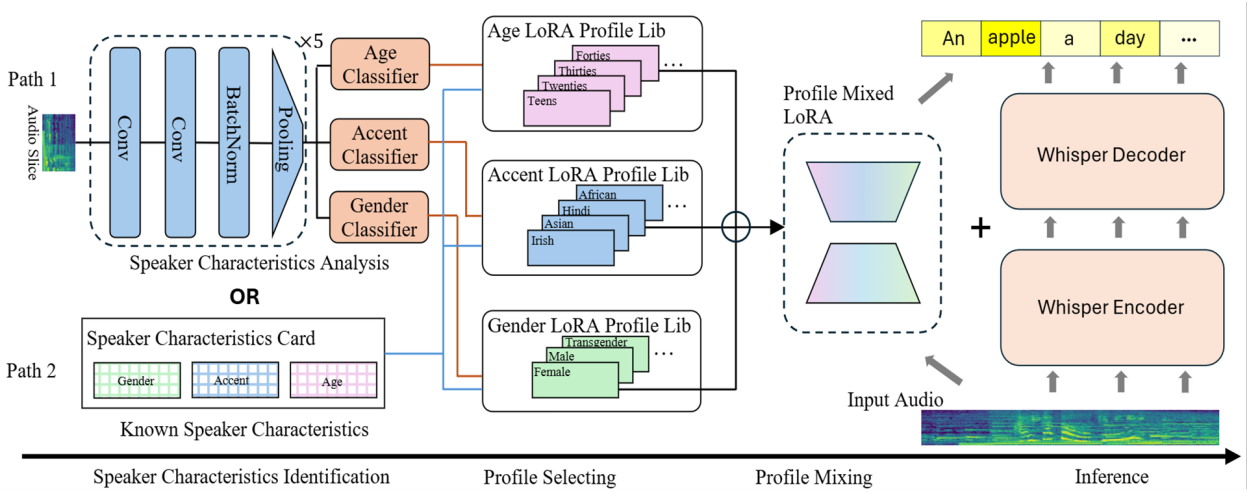

Figure 1: PI-Whisper leverages multiple LoRA profile libraries and dynamic LoRA merging of LoRA profiles to adjust ASR towards the speaker's characteristics. When the speakers' characteristics are not known, PI-Whisper employs a multi-head classifier to infer the characteristics from audio samples.

2 Background & Related Works

To contextualize the deployment and system design of PI-Whisper, it is necessary to outline the core motivations for edge based speech recognition and the architectural challenges associated with deploying foundational models on constrained hardware.

2.1 Motivation for Edge ASR Deployment

The migration of ASR from centralized cloud servers to edge devices is driven by several critical operational benefits. Processing speech data locally immediately resolves inherent privacy concerns by avoiding the transmission of sensitive audio over networks [2]. Furthermore, localized on-device inference entirely eliminates the communication delays associated with cloud architectures, unlocking the ultra-low latency required for real time applications [3].

2.2 Hardware Constraints and Foundational Models

While edge deployment solves latency and privacy issues, it introduces severe resource constraints, as edge hardware possesses limited processing power, memory, and energy [4]. Foundational models, such as OpenAI's Whisper [5], achieve state-of-the-art accuracy through massive parameter counts, which historically made them unsuitable for edge environments. PI-Whisper addresses this by extending the Whisper-tiny base model (37.7M parameters) for seamless edge deployment on devices like the NVIDIA Jetson and Raspberry Pi. To guarantee optimal performance, PI-Whisper operates on a distributed architecture that combines strict task-scoped CUDA memory controls with zero-copy MJPEG buffering to achieve ultra-low streaming latency.

2.3 Parameter-Efficient Fine-Tuning (PEFT) and LoRA

A primary challenge in edge ASR is adaptivity, the ability to perform real-time adjustments to a speaker's unique characteristics such as accent, gender, and age. Standard methodologies require retraining entire models to accommodate this variance, which lacks incrementality and is computationally unfeasible on the

edge [6]. To achieve this, PI-Whisper leverages Low-Rank Adaptation (LoRA) [7]. PEFT methods adapt large pretrained models by updating a small number of additional parameters, rather than retraining the full model. LoRA is a widely used PEFT [8] technique that inserts trainable low-rank update matrices into selected model layers while keeping the original model weights frozen. PI-Whisper applies this idea to Whisper-tiny by training characteristic-specific LoRA profiles for speaker attributes such as accent, gender, and age. Instead of maintaining separate fully fine-tuned ASR models, PI-Whisper keeps a shared base model and dynamically selects or merges LoRA profiles according to known or inferred speaker characteristics. This design allows the system to adapt to diverse speaker groups with lower training and deployment overhead, while supporting incremental updates when new speaker characteristics are introduced.

2.4 Cloud-Reliant vs. Edge-Native Architectures

Traditional high-accuracy ASR systems rely heavily on massive server-side compute arrays [9]. While this allows for the deployment of highly parameterized models, it introduces network latency and limits offline usability. Conversely, native edge systems have historically traded accuracy for speed. PI-Whisper bridges this gap, introducing a novel, modular ASR framework tailored for edge devices [1].

2.5 Algorithmic Fairness and Inclusivity

A critical area of related work focuses on ensuring equitable ASR performance across diverse demographics, as many foundational models exhibit performance gaps across different populations [10, 11]. When evaluating fairness based on Age, the Base Model exhibits a Statistical Parity Difference (SPD) of 0.083 and a Disparate Impact Ratio (DIR) of 1.929. While mitigation strategies like "One For All" offer some improvement, PI-Whisper's dynamic, characteristic-aware adaptation yields superior fairness metrics. By simultaneously injecting age, accent, and gender profiles, PI-Whisper(All) reduces the SPD to an optimal 0.044 and the DIR to 1.626.

2.6 Performance and Accuracy Baselines

Prior research has explored various mechanisms for adapting ASR models to reduce Word Error Rate (WER), establishing baselines using architectures like Trans, I-vector, and Conformer [12, 13]. PI-Whisper was rigorously evaluated against these baselines using the L2-Arctic dataset, featuring non-native English speakers [14], and CommonVoice, containing diverse accents and demographics [15]. Compared to standard approaches that exhibit higher error rates, the dynamic profile mixing utilized by PI-Whisper achieves up to a 13.7% relative WER reduction. Crucially, it manages this significant accuracy gain while adding only 0.18 seconds of inference time per sample, cementing its viability for real-world execution.

3 Methodology

This section outlines the system design and implementation of PI-Whisper. The implementation integrates a responsive frontend, an asynchronous backend task orchestrator, and an intensive machine learning inference pipeline optimized for Nvidia Jetson. Fig. 2 shows the complete design and different sections involved in building the complete platform.

3.1 High Level Architecture Overview

PI-Whisper is designed as a full-stack, event-driven application to ensure the user interface remains responsive while the system performs heavy computational tasks in the background. The architecture is logically divided into four primary layers:

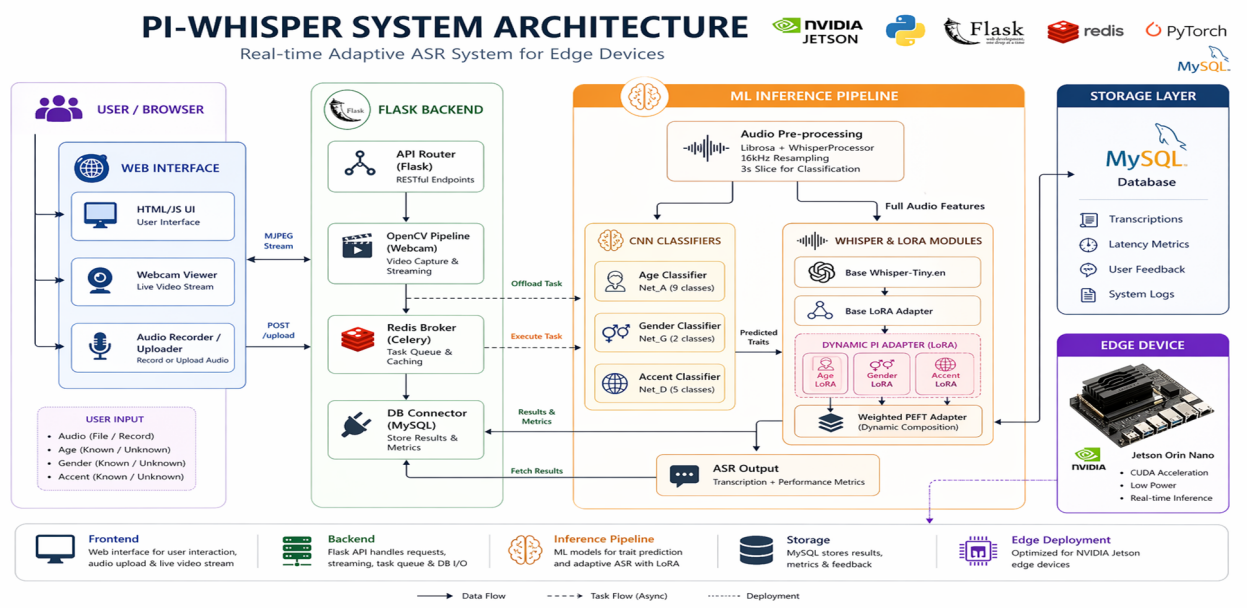

Figure 2: System Architecture of PI-Whisper Website.

1. **Frontend UI:** A web-based interface for data collection and real-time streaming.
2. **Backend Server:** A Flask-based API that routes requests and manages hardware streams.
3. **Task Queue:** A Celery and Redis broker system for handling asynchronous inference jobs.
4. **Inference Pipeline:** A PyTorch-based execution environment managing demographic classification and dynamic LoRA composition.

3.2 User Interface and Data Ingestion

The frontend serves as the primary user touchpoint and is built with standard HTML, CSS, and JavaScript. It is engineered to handle multimodal inputs:

1. **Audio Capture:** The interface supports direct .wav file uploads and browser-recorded audio converted to WAV before upload. The MediaRecorder API captures user speech, which is then normalized using a WAV-encoder strategy prior to backend submission.
2. **Parameter Selection:** Users can manually input their demographic characteristics (Age, Gender, Accent). If these fields are set to ”unknown,” the UI signals the backend to trigger the automated classification pipeline.
3. **Real-Time Webcam Integration:** The frontend continuously consumes an MJPEG video stream directly from the backend via a dedicated /video_feed endpoint, with /video_single_frame used for the low-latency modal stream. This stream is mainly showing the Jetson Orin that we are using to deploy both the website and backend.

3.3 Backend Orchestration and Asynchronous Processing

The backend layer is built on the Flask framework, serving as a the central coordinator between the user interface, hardware peripherals, and the machine learning pipeline.

1. **API and Stream Management:** Flask manages all RESTful routings (/upload, /task-status/< task_id>, /results/< session_id>, /submit-feedback, /video_feed, /video_single_frame). It also directly interfaces with the edge device's camera hardware via cv2.VideoCapture(), supporting both hardware-accelerated CSI (GStreamer) and standard USB webcams (V4L2).

2. **Celery and Redis Integration:** Audio transcription and demographic classification require multi-second execution times, which would traditionally block the main web thread. To circumvent this, the Flask application offloads processing tasks to Celery workers. Redis acts as the asynchronous message broker and caching system, queuing jobs and returning results without interrupting the video stream or UI responsiveness.

3. **Data Persistence:** A relational database connection (utilizing MySQL) logs output transcriptions, inference latency metrics, and user feedback, establishing a robust foundation for ongoing performance evaluation.

3.4 Machine Learning and Inference Pipeline

The core of the Pi-Whisper is in its machine learning pipeline, implemented utilizing pytorch, transformers and peft. There were several challenges that arose while setting up the libraries on Nvidia Jetson. It does not support all the versions for some of the packages. In order for the hardware accelerator to work properly, we had to download specific versions from source code available online and make the cuda work properly. The pipeline is highly modular, executing sequentially upon receiving a task from the Celery worker.

1. **Audio Pre-Processing:** Incoming audio payloads are immediately stripped and down-sampled to 16kHz utilizing the librosa library. A WhisperProcessor handles the entire sequence for the primary ASR task. During preprocessing, the same processed Whisper input features are also sliced to [:,:,:300] for demographic classification.

2. **Trait Classifiers (CNNs):** If demographic parameters are not explicitly provided by the user, the 3-second audio slice is routed through three separate 12-layer Convolutional Neural Networks (CNNs):

   (a) Net_G: Predicts Gender across 2 distinct classes.

   (b) Net_D: Predicts Accent across 5 classes (e.g., Australian, US, England, Canadian, Scottish).

   (c) Net_A: Predicts Age across 9 distinct strata ranging from teens to nineties.

3. **Dynamic LoRA Ensembling:** PI-Whisper evaluates the audio against multiple configurations sequentially to benchmark performance. It tracks the Base Whisper (OpenAI's whisper-tiny.en) and a statically trained baseline adapter. For the personalized PI-Whisper output, the pipeline retrieves three discrete LoRA adapters corresponding to the predicted (or provided) traits. These modules are dynamically pooled using a Parameter-Efficient Fine-Tuning (PEFT) adapter via add_weighted_adapter. To preserve the complete operational capacity of each characteristic transformation, the system utilizes an equal-weight concatenation logic (assigning a combining weight of 1.0 to each individual profile) to form a unified composite adapter. This dynamically merged profile processes the Mel Spectrogram to yield the final, personalized transcription.

3.5 Edge Hardware Integrity and Optimization

Deploying this sophisticated pipeline on constrained edge environments requires strict system-level optimizations.

1. **Task Scoped GPU execution:** The pipeline is explicitly written to leverage single-device CUDA pipelines (cuda:0). To prevent memory fragmentation and ensure stability, ML models are initialized using task-scoped boundaries inside each Celery inference task, so model/CUDA initialization happens in the worker task rather than as a global Flask import. This guarantees that CUDA runtime initializations do not break across asynchronous worker bounds.

2. **Latency Control:** Tailored specifically for NVIDIA Jetson hardware, the system implements a latest-frame buffer with single frame V4L2/GStreamer buffering, optimized JPEG/MJPEG encoding, and no cache HTTP headers. This ensures that the continuous video feed maintains ultra-low latency even while the backend compute cycles are heavily saturated by the PyTorch audio processing tasks.

4 Results

4.1 Evaluation Scope

This project builds upon the PI-Whisper research framework, which proposes adaptive automatic speech recognition using speaker-specific Low-Rank Adaptation (LoRA) profiles. While the original PI-Whisper work focuses on model-level improvements in transcription accuracy and fairness, this project investigates a complementary research question: whether the PI-Whisper methodology can be transformed into a complete, interactive, edge-deployable system.

The evaluation therefore considers two categories of results. First, the algorithmic performance reported in the original PI-Whisper research is used as the model-level baseline for understanding the benefits of dynamic LoRA adaptation. Second, this project evaluates the practical system-level outcome of implementing PI-Whisper as a web-based platform deployed on NVIDIA Jetson hardware. The main contribution of this work lies in validating the feasibility of integrating frontend interaction, asynchronous backend processing, demographic classification, dynamic adapter selection, database persistence, and edge deployment into one operational system.

4.2 Research Baseline: Accuracy and Fairness Results

The original PI-Whisper research demonstrates that dynamic profile-aware adaptation can improve transcription accuracy compared with standard ASR baselines. Using datasets such as L2-Arctic and Common-Voice, Fig. 3 reports up to a 13.7% relative reduction in Word Error Rate (WER). Standard baselines such as Trans and I-vector produced higher WER values, while context-aware baselines provided only limited improvement.

In the reported PI-Whisper results, accent-based adaptation reduced WER to 11.04, while the fully personalized configuration combining age, gender, and accent achieved the best WER of 9.62. These results support the core research motivation behind PI-Whisper: speaker-aware adaptation can improve ASR performance for diverse users without retraining the entire Whisper model.

Table 1 also reports fairness improvements across demographic groups. For age-based fairness, the base model produced an SPD of 0.083 and a DIR of 1.929. The fully personalized PI-Whisper configuration reduced these values to an SPD of 0.044 and a DIR of 1.626, indicating reduced performance disparity across speaker groups. These findings provide the algorithmic foundation for the system implemented in this project. In Fig. 4, the authors showed tradeoff between inference time and WER on both hardware under both settings (known and Inferred).

Table 1: Fairness comparison based on Age

Model	SPD	DIR
Base Model	0.083	1.929
One For All	0.058	1.795
PI-Whisper(Age)	0.057	1.698
PI-Whisper(ALL)	**0.044**	**1.626**

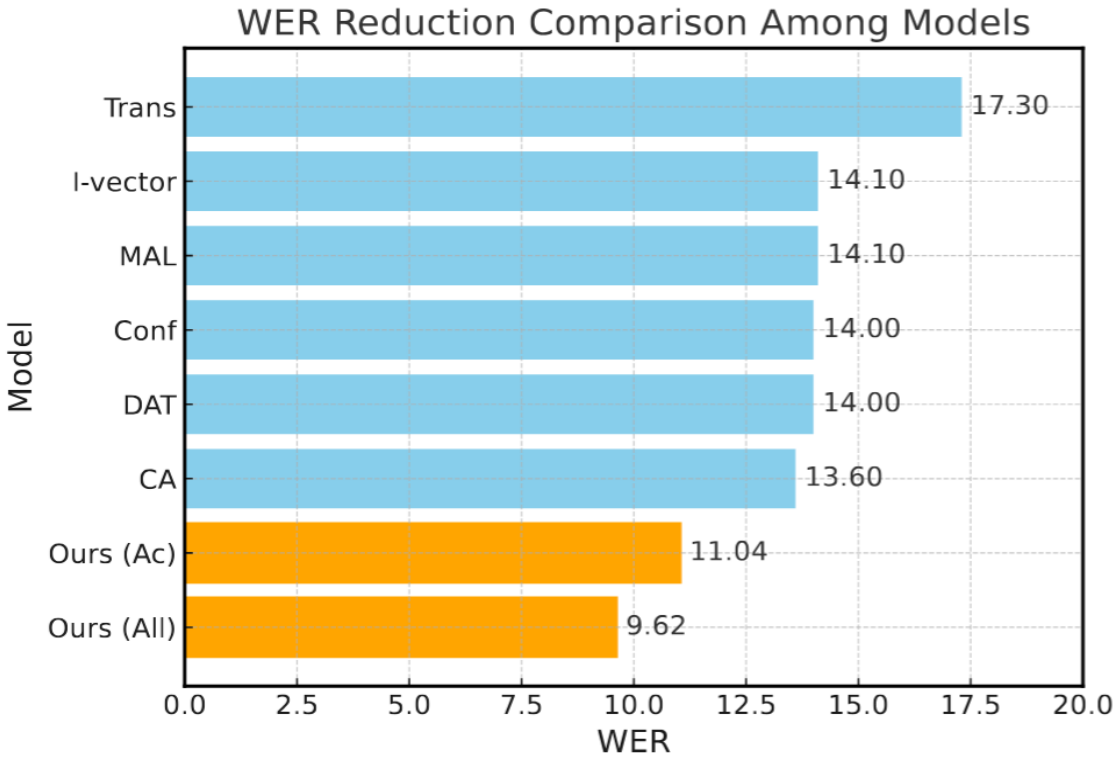

Figure 3: Comparison of WER across various models. PI-Whisper achieves lower WER compared to other baselines.

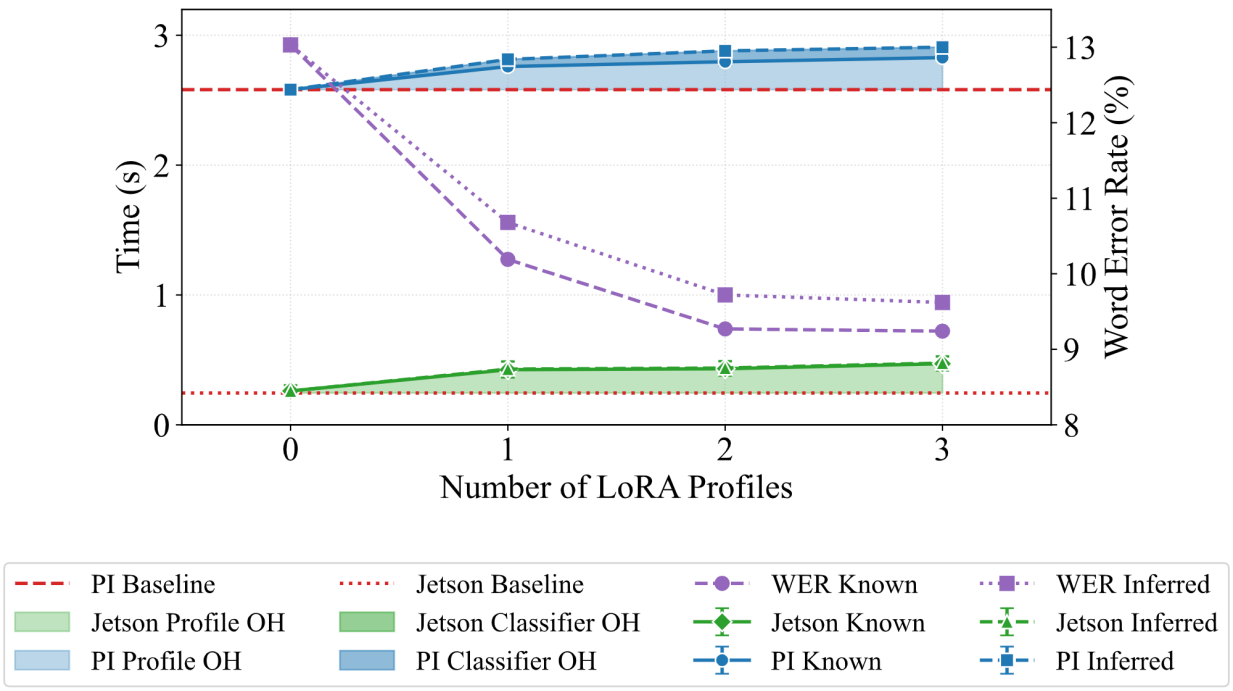

Figure 4: Impact of LoRA profiles on both WER and Inference time.

4.3 End-to-End System Integration and Feasibility Validation

The primary result of this project is the successful implementation of PI-Whisper as an interactive edge-based web platform. The system allows users to upload or record speech, manually provide demographic traits, or mark traits as unknown so that the system can automatically predict them. The backend then uses the predicted or user-provided age, gender, and accent values to select the appropriate LoRA adapters and generate personalized transcription output.

This demonstrates that PI-Whisper can be moved beyond an offline research pipeline into a usable, real-time application workflow. The implemented platform integrates three major transcription pathways: the base Whisper model, a one-LoRA baseline, and the dynamically personalized PI-Whisper configuration. This allows the system to compare standard and adapted ASR behavior within a single deployed interface.

A key research outcome of this implementation is the validation of end-to-end system feasibility. The project shows that demographic classification, dynamic adapter composition, transcription generation, latency measurement, result storage, and feedback collection can be coordinated within one deployable architecture on edge hardware.

4.4 Asynchronous Inference and User Responsiveness

A major challenge in deploying PI-Whisper is that ASR inference and adapter-based model execution are computationally expensive. Running these tasks directly inside the web request cycle would block the interface and reduce usability. To address this, the system uses an asynchronous backend architecture based on Flask, Celery, and Redis.

When a user submits audio, the Flask backend stores the file and dispatches the inference task to a Cel-

ery worker. The frontend then polls the backend for task completion instead of waiting synchronously. This design allows the user interface to remain responsive while the system performs demographic classification and transcription in the background.

This result is significant from a systems perspective because it shows that adaptive ASR pipelines can be integrated into interactive applications without forcing users to wait on a frozen web request. The separation between request handling and model execution makes the system more practical for real-world deployment on resource-constrained devices.

4.5 Edge Deployment Results on NVIDIA Jetson

The project successfully deploys the PI-Whisper platform on NVIDIA Jetson hardware using CUDA-enabled PyTorch. This validates the practical feasibility of running an adaptive Whisper-based ASR system on an edge device rather than relying on cloud inference. The deployed system includes the web frontend, Flask backend, Celery worker, Redis broker, MySQL database connection, webcam streaming, and PyTorch inference pipeline. Fig. 5 shows the website interface created to run the inferences.

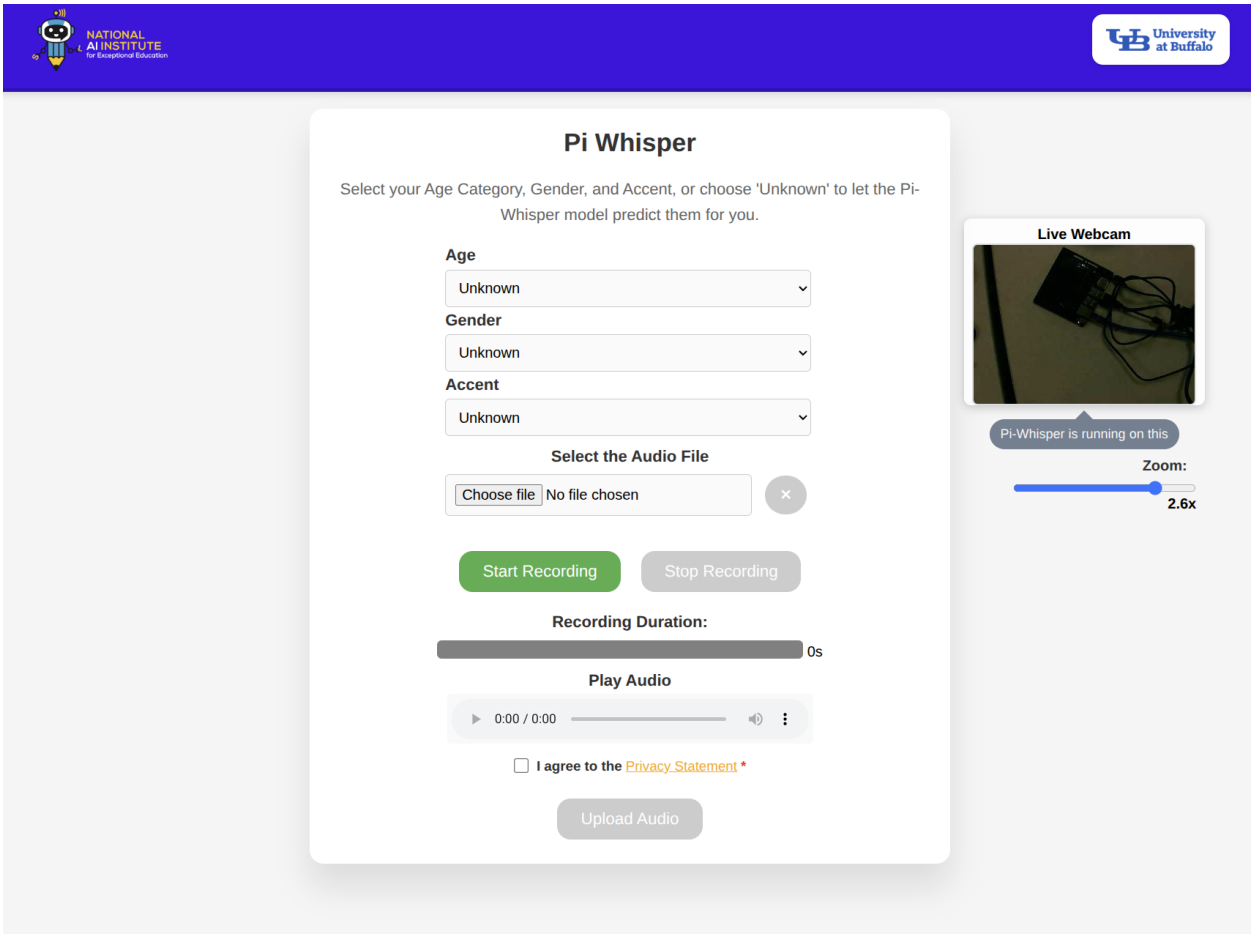

Figure 5: Implemented PI-Whisper web interface running on Jetson.

The deployment also uses process management to improve reliability. The Flask server and Celery worker are managed as separate processes, allowing the backend and inference worker to restart independently if

needed. This is important for edge environments, where long-running applications must remain stable without constant manual supervision. Fig. 6 shows the results obtained using the website interface when an audio was recorded on the website and processed to perform transcription.

The implementation confirms that Jetson deployment requires careful dependency management, particularly for CUDA-compatible versions of PyTorch, Transformers, PEFT, and audio-processing libraries. These constraints are an important practical finding of this project because they show that deploying adaptive ASR on edge hardware is not only a model-design problem, but also a system-integration challenge.

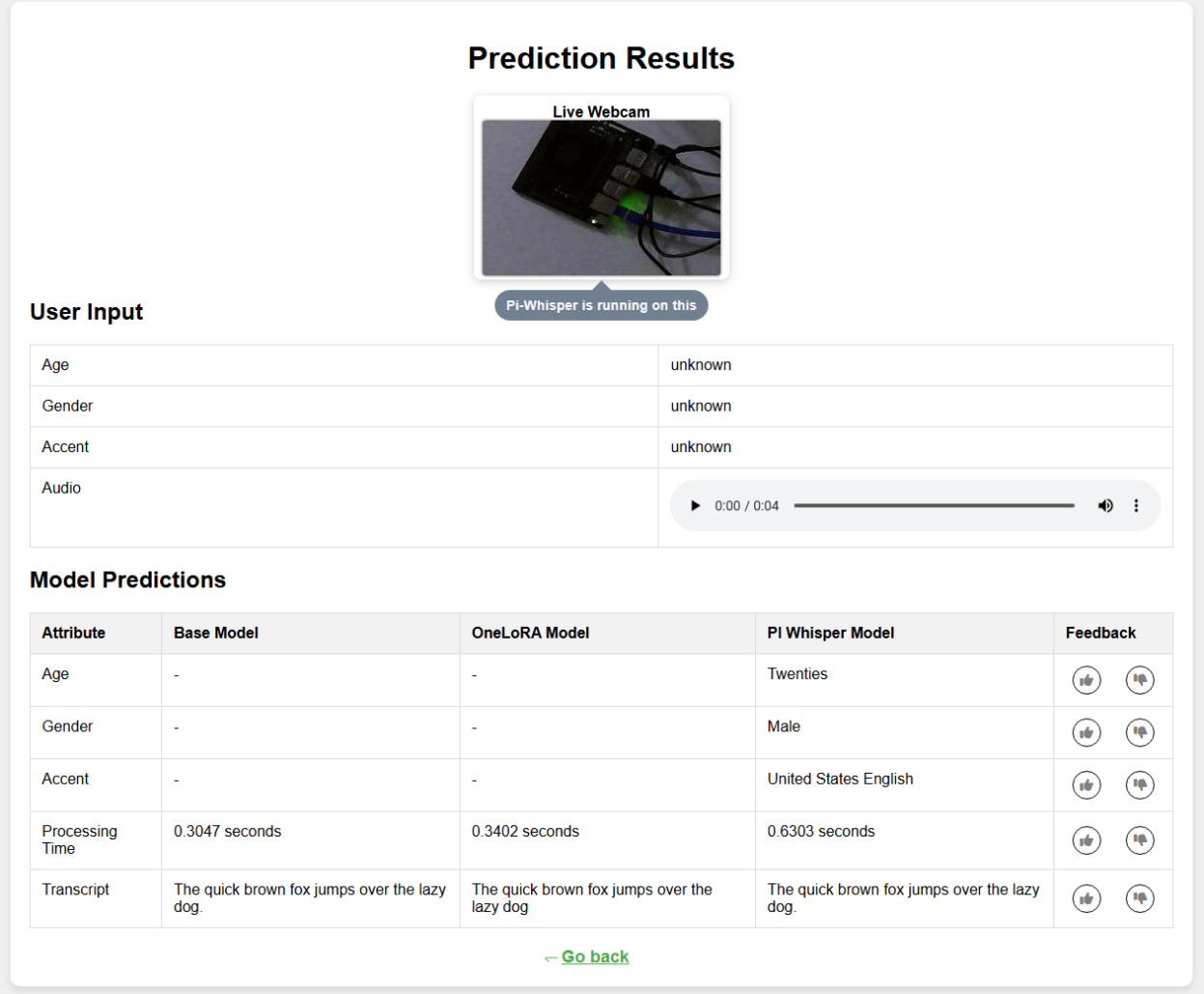

Figure 6: Results displayed for real time transcription.

5 Conclusion

This project implemented and deployed PI-Whisper as an interactive edge-based speech recognition platform on NVIDIA Jetson hardware. Building on the original PI-Whisper research, the work focused on converting the adaptive ASR methodology into a practical system with a web frontend, Flask backend, Celery/Redis asynchronous processing, MySQL persistence, CUDA-enabled inference, demographic classification, and dynamic LoRA-based transcription.

The main contribution of this project is the system-level validation of PI-Whisper in a real deployment setting. The platform allows users to upload or record speech, provide or omit demographic traits, compare transcription outputs, and submit feedback through a unified interface. This demonstrates that speaker-aware ASR can be integrated into a usable edge application rather than remaining only an offline research pipeline.

The project also highlights practical deployment challenges, including Jetson-compatible dependencies, GPU memory management, asynchronous inference, and low-latency interaction design. Overall, this work shows that personalized, privacy-preserving ASR can be deployed locally on edge hardware while maintaining an interactive user experience.

References

[1] A. Nassereldine, D. Liu, C. Xu, R. Qin, Y. Shi, and J. Xiong, "Pi-whisper: Designing an adaptive and incremental automatic speech recognition system for edge devices," 2024.

[2] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," 2019.

[3] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," 2020.

[4] H. Cai, J. Lin, Y. Lin, Z. Liu, H. Tang, H. Wang, L. Zhu, and S. Han, "Enable deep learning on mobile devices: Methods, systems, and applications," *ACM Transactions on Design Automation of Electronic Systems*, vol. 27, p. 1–50, Mar. 2022.

[5] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," 2022.

[6] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. de Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," 2019.

[7] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021.

[8] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, B. Bossan, and M. Tietz, "PEFT: State-of-the-art parameter-efficient fine-tuning methods." `https://github.com/huggingface/peft`, 2022.

[9] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," 2021.

[10] A. Koenecke, A. Nam, E. Lake, J. Nudell, M. Quartey, Z. Mengesha, C. Toups, J. R. Rickford, D. Jurafsky, and S. Goel, "Racial disparities in automated speech recognition," *Proceedings of the National Academy of Sciences*, vol. 117, no. 14, pp. 7684–7689, 2020.

[11] Y. Meng *et al.*, "On the fairness of speech recognition models," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021.

[12] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," 2020.

[13] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented transformer for speech recognition," 2020.

[14] G. Zhao, S. Sonsaat, A. Silpachai, I. Lucic, E. Chukharev-Hudilainen, J. Levis, and R. Gutierrez-Osuna, "L2-ARCTIC: A Non-native English Speech Corpus," in *Interspeech 2018*, pp. 2783–2787, 2018.

[15] R. Ardila, M. Branson, K. Davis, M. Henretty, M. Kohler, J. Meyer, R. Morais, L. Saunders, F. M. Tyers, and G. Weber, "Common voice: A massively-multilingual speech corpus," 2020.